#include "internal/error.h"

#include "internal/thrift.h"

#include <map>

#include <twml/ThriftWriter.h>

#include <twml/DataRecordWriter.h>

#include <twml/io/IOError.h>

#include <unordered\_set>

using namespace twml::io;

namespace twml {

void DataRecordWriter::writeBinary(twml::DataRecord &record) {

const DataRecord::BinaryFeatures bin\_features = record.getBinary();

if (bin\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_SET, DR\_BINARY);

m\_thrift\_writer.writeListHeader(TTYPE\_I64, bin\_features.size());

for (const auto &it : bin\_features) {

m\_thrift\_writer.writeInt64(it);

}

}

}

void DataRecordWriter::writeContinuous(twml::DataRecord &record) {

const DataRecord::ContinuousFeatures cont\_features = record.getContinuous();

if (cont\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_CONTINUOUS);

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_DOUBLE, cont\_features.size());

for (const auto &it : cont\_features) {

m\_thrift\_writer.writeInt64(it.first);

m\_thrift\_writer.writeDouble(it.second);

}

}

}

void DataRecordWriter::writeDiscrete(twml::DataRecord &record) {

const DataRecord::DiscreteFeatures disc\_features = record.getDiscrete();

if (disc\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_DISCRETE);

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_I64, disc\_features.size());

for (const auto &it : disc\_features) {

m\_thrift\_writer.writeInt64(it.first);

m\_thrift\_writer.writeInt64(it.second);

}

}

}

void DataRecordWriter::writeString(twml::DataRecord &record) {

const DataRecord::StringFeatures str\_features = record.getString();

if (str\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_STRING);

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_STRING, str\_features.size());

for (const auto &it : str\_features) {

m\_thrift\_writer.writeInt64(it.first);

m\_thrift\_writer.writeString(it.second);

}

}

}

// convert from internal representation list<(i64, string)>

// to Thrift representation map<i64, set<string>>

void DataRecordWriter::writeSparseBinaryFeatures(twml::DataRecord &record) {

const DataRecord::SparseBinaryFeatures sp\_bin\_features = record.getSparseBinary();

// write map<i64, set<string>> as Thrift

if (sp\_bin\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_SPARSE\_BINARY);

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_SET, sp\_bin\_features.size());

for (auto key\_vals : sp\_bin\_features) {

m\_thrift\_writer.writeInt64(key\_vals.first);

m\_thrift\_writer.writeListHeader(TTYPE\_STRING, key\_vals.second.size());

for (auto name : key\_vals.second)

m\_thrift\_writer.writeString(name);

}

}

}

// convert from internal representation list<(i64, string, double)>

// to Thrift representation map<i64, map<string, double>>

void DataRecordWriter::writeSparseContinuousFeatures(twml::DataRecord &record) {

const DataRecord::SparseContinuousFeatures sp\_cont\_features = record.getSparseContinuous();

// write map<i64, map<string, double>> as Thrift

if (sp\_cont\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_SPARSE\_CONTINUOUS);

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_MAP, sp\_cont\_features.size());

for (auto key\_vals : sp\_cont\_features) {

m\_thrift\_writer.writeInt64(key\_vals.first);

if (key\_vals.second.size() == 0)

throw IOError(IOError::MALFORMED\_MEMORY\_RECORD);

m\_thrift\_writer.writeMapHeader(TTYPE\_STRING, TTYPE\_DOUBLE, key\_vals.second.size());

for (auto map\_str\_double : key\_vals.second) {

m\_thrift\_writer.writeString(map\_str\_double.first);

m\_thrift\_writer.writeDouble(map\_str\_double.second);

}

}

}

}

void DataRecordWriter::writeBlobFeatures(twml::DataRecord &record) {

const DataRecord::BlobFeatures blob\_features = record.getBlob();

if (blob\_features.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_BLOB);

m\_thrift\_writer.writeMapHeader(TTYPE\_I64, TTYPE\_STRING, blob\_features.size());

for (const auto &it : blob\_features) {

m\_thrift\_writer.writeInt64(it.first);

std::vector<uint8\_t> value = it.second;

m\_thrift\_writer.writeBinary(value.data(), value.size());

}

}

}

void DataRecordWriter::writeDenseTensors(twml::DataRecord &record) {

TensorRecord::RawTensors raw\_tensors = record.getRawTensors();

if (raw\_tensors.size() > 0) {

m\_thrift\_writer.writeStructFieldHeader(TTYPE\_MAP, DR\_GENERAL\_TENSOR);

m\_tensor\_writer.write(record);

}

}

TWMLAPI uint32\_t DataRecordWriter::getRecordsWritten() {

return m\_records\_written;

}

TWMLAPI uint64\_t DataRecordWriter::write(twml::DataRecord &record) {

uint64\_t bytes\_written\_before = m\_thrift\_writer.getBytesWritten();

writeBinary(record);

writeContinuous(record);

writeDiscrete(record);

writeString(record);

writeSparseBinaryFeatures(record);

writeSparseContinuousFeatures(record);

writeBlobFeatures(record);

writeDenseTensors(record);

// TODO add sparse tensor field

m\_thrift\_writer.writeStructStop();

m\_records\_written++;

return m\_thrift\_writer.getBytesWritten() - bytes\_written\_before;

}

} // namespace twml